intel®

AP-374

APPLICATION NOTE

Flash Memory Write Protection Techniques

BRIAN DIPERT
SENIOR TECHNICAL MARKETING ENGINEER

December 1995

Order Number: 292123-001

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

FLASH MEMORY WRITE PROTECTION TECHNIQUES

CONTENTS

PAGE

CONTENTS

PAGE

1.0 INTRODUCTION

Flash memory's combination of nonvolatility and easy in-system updateability are key attributes driving its adoption into today's system designs. However, this flexibility also brings with it the responsibility (for hardware and software engineers) to ensure that writes to flash memory occur *only when intended.* This is especially important for those who are accustomed to designing with various ROM (nonvolatile but non-updateable) and RAM (updateable but volatile) memories.

This application note discusses techniques for proactively designing systems to prevent unintentional writes to flash memory. These design techniques are by no means complex or costly, but their implementation is crucial to ensuring reliable operation through system lifetime. For more information on the devices and specifications discussed in this document, please consult specific flash memory datasheets.

2.0 WHY IS WRITE PROTECTION IMPORTANT?

Let's begin by identifying the key characteristics of two generic memory technologies: ROM (Read-Only-Memory) and RAM (Random-Access-Memory). Flash memory combines many of the capabilities of both in one solution. Therefore, it is often being utilized to replace ROM and/or RAM in new designs. At a minimum, flash memory's status as a relatively new technology means that many engineers are moving to it from the familiarity of a ROM/RAM knowledge base.

RAM is fully alterable on a bit-by-bit basis, and the mechanism for writing to it is established and well understood. RAM is in-system updateable, yet it is volatile. This means that when a RAM memory loses power, it also loses its data. RAM is guaranteed *not* to contain valid information on powerup.

ROM offers the advantage of nonvolatility, i.e. when power is removed from the device, the information stored inside is retained. However, ROM is not in-system updateable. Once the information is initially put into the device, it is permanent and unchangeable. To replace the information, you have to physically remove/replace the device itself.

Traditional system memory architectures often included both ROM (nonvolatile but non-updateable) and RAM (volatile but in-system updateable). The new model for system design retains some RAM for temporary data storage, but replaces the rest of RAM and ROM with flash memory. Being both nonvolatile and in-system updateable, flash memory encompasses the strengths of both RAM and ROM, offering new system architecture possibilities. However, whereas in the past RAM was guaranteed to be invalid on system powerup and ROM was guaranteed to be unalterable, the same cannot be said for flash memory.

Any alteration of flash memory contents (whether planned or unintended) is permanent regardless of system power transitions, until the data is again modified. As we'll see later, command writes to flash memory can also put it in modes where it outputs something other than array data, a non-permanent but still undesirable condition when not intended. This means that the system hardware and software must ensure that flash memory is written only when specifically desired, to ensure a predictable system environment. The following sections will discuss how this can be accomplished.

3.0 SYSTEM WRITES WITH BULK-ERASE FLASH MEMORIES

First-generation bulk erase flash memories from Intel Corporation are shown in Figure 1. These devices automatically power up in a "Read Array" mode in which they output array data when read. Transitions to alternate modes occur by writing commands to the flash memory.

Device	Density
28F256A	32 Kbytes (x8)
28F512	64 Kbytes (x8)
28F010	128 Kbytes (x8)
28F020	256 Kbytes (x8)

Figure 1. Intel Corporation Bulk-Erase Flash Memories

Bulk-erase flash memories include several forms of "protection" to guard against unintended writes. Writes with V_{PP} (the program/erase voltage) at V_{PPL} (0V to 6.5V) are disregarded by the flash memory. Similarly, write attempts with V_{CC} at or below V_{LKO} (2.5V on most devices) are ignored. Finally, these devices require multi-byte command sequences to initiate internal program or erase algorithms. Note, however, that while the erase command sequence (shown in Figure 2) requires both the proper Erase Setup and Erase Confirm commands, the program sequence (Figure 3) relies only on the valid Program Setup command. The second command in the latter sequence can have any value, and is interpreted as data to be programmed. This means that if the flash memory receives an unintended Program Setup command, the very next write to the device (intended or not) will be interpreted as program data and initiate an internal program event (if V_{PP} is above V_{PPL}).

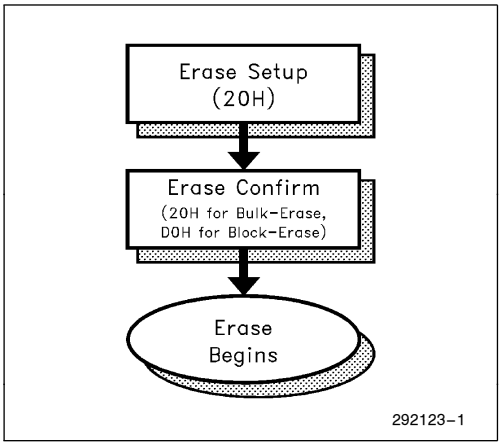

Figure 2. Flash Memory Erase Command Sequence (Simplified)

Beyond the program and erase sequences, the Read Intelligent Identifier Codes command will, when written to the flash memory, put it in a mode where it outputs device signature IDs instead of array information when read.

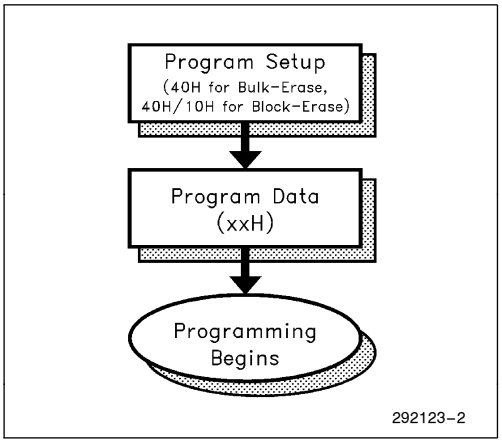

Figure 3. Flash Memory Programming Command Sequence (Simplified)

4.0 SYSTEM WRITES WITH BOOT BLOCK AND FlashFile™ MEMORIES

Second-generation block-erase Boot Block and FlashFile memories from Intel Corporation are shown in Figure 4. They function similarly to the bulk-erase devices described earlier, with a few key enhancements. As before, these devices automatically power up in "Read Array" mode, and transition to alternate modes via command writes.

BOOT BLOCK ARCHITECTURE	
Device	**Density**
28F001BX	128 Kbytes (x8)
28F200BX	256 Kbytes (x16)
28F002BX	256 Kbytes (x8)
28F400BX	512 Kbytes (x16)
28F004BX	512 Kbytes (x8)
FlashFile™ ARCHITECTURE	
Device	**Density**
28F008SA	1 Mbyte (x8)

Figure 4. Intel Corporation Block-Erase Flash Memories

For full access to the flash memory Status Register, as well as for enhanced interface to internal device identifiers, these block-erase flash memories will accept commands written to them regardless of V_{PP} voltage, as long as V_{CC} is above V_{LKO}. Program and erase algorithms initiated by command sequences will terminate with Status Register error indication and unaltered array data, if V_{PP} is at V_{PPL}. However, regardless of V_{PP} level, the device will still transition to a "Read Status Register" mode after program/erase command sequences are written. In this case, it will output data that the system, if the write was unintended, will not expect. The same multi-byte command sequences (shown in Figures 2 and 3) are used as in bulk-erase flash memories.

Boot Block and FlashFile memories provide commands (in addition to the program and erase sequences) which transition the memory to alternate modes, outputting data other than array information for subsequent reads. In this respect, they are similar to bulk-erase flash memory discussed earlier. These commands are Intelligent Identifier and Read Status Register.

Block-erase devices include a hardware input called RP# (or Reset/Powerdown). Among its many uses, this pin acts as a "master on/off switch" to completely disable the flash memory and lock all other control inputs. RP# is extremely effective at blocking unintended writes during system power transitions. This technique will be covered in detail, in a few paragraphs.

5.0 PREVENTING UNINTENDED WRITES DURING NORMAL SYSTEM OPERATION

Preventing unintended writes to flash memory during normal system operation is a routine part of debugging a new design, and a common concern for any "writeable" device on the processor interface. Any combina-

tion of active chip select (CE#) and active write enable (WE#) has the potential of being decoded by the flash memory as a valid write attempt. One common culprit in these situations is the chip select decoder logic (PAL, etc.) between the processor and external devices. As addresses propogate through this logic at the beginning of an access cycle, or in the undefined address state between accesses, spurious chip selects of indeterminate duration can be generated. System hardware should ensure that at these times, WE# to flash memory stays at a logic "1" and doesn't transition low.

Some concern has also been expressed in the past about unintended writes in certain "open" systems such as the personal computer. In these environments, the type and function of software run on the machine is beyond the control of the computer manufacturer, who must accordingly design his/her hardware. For example, a third-party software utility may write to flash memory assuming DRAM at that location. More malicious, of course, is the case of the computer virus. Fortunately, in cases like this, hardware design to prevent unintended writes is fairly simple.

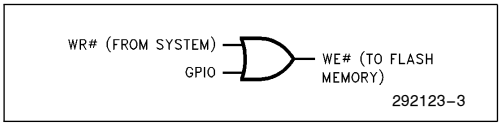

Figure 5. WE# Gating

Figure 5 shows one means of clarifying the WE# signal. When flash memory is used for BIOS storage, for example, the manufacturer's update utility is the only software that should be writing to the device. By toggling the general purpose I/O line (whose default state is, of course, "disabled"), the update utility can control whether writes from the system are blocked or allowed to pass to the flash memory. This type of WE# clarifying function is integrated in the Intel386™SL and Intel486™SL Microprocessor Supersets. ASICs integrating motherboard functions should also be designed to include such logic.

One other method for preventing flash memory alteration is by controlling (or "switching") the V_{PP} voltage, turning it on to V_{PPH} only when desired for system update. Many 12V converters and power supplies integrate this on/off function as shown in Figure 6, or it can be provided by an external FET. This approach will be used again in the next section on write protection during system power transitions. Note, however, that although it prevents actual flash memory data alteration, V_{PP} control is insufficient to keep block-erase flash memories from transitioning to alternate data output modes by unintended writes.

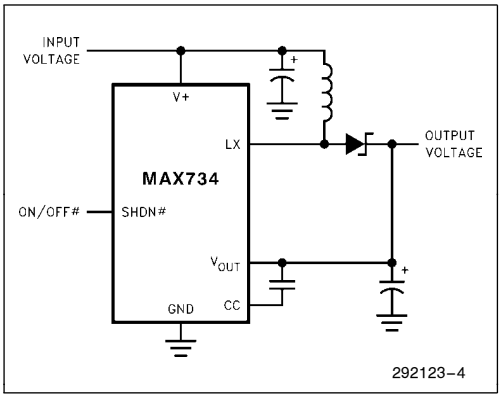

Figure 6. V_{PP} 12V Converter with Integrated Switch (Example)

In a traditional "closed" system, the software directing the hardware is totally under control of the system manufacturer. No additional effort should be needed (after the initial prototype hardware and software debugging) to protect the flash memory from unintended writes during normal system operation. Write control during system powerup and powerdown also requires attention, however; a topic covered next.

6.0 PREVENTING UNINTENDED WRITES DURING SYSTEM RESET AND POWERUP/ POWERDOWN

System powerup and/or powerdown offer the greatest potential for unintended writes in flash memory-based system designs. As mentioned earlier, similar potential also exists for other "nonvolatile/rewriteable" memory technologies, such as EEPROM and battery-backed SRAM. Several reasons for this are listed below.

- When a system begins to power up, all logic outputs are at 0V. This is also the "enable" condition for flash memory CE# and WE# inputs.
- Logic devices have specified, documented and guaranteed operation only at a specific supply voltage range (typically 5V ±10% or 3.3V ±0.3V). Operation beyond this voltage range is not guaranteed and may not be consistent. Specifically, device output behaviour is typically undefined.
- Similarly, logic operation is sometimes undefined and erratic when devices are being reset. For example, MCS-186 embedded processors, when reset, tristate their WR# (write enable) outputs, which will then typically drift toward 0V (or "enabled", to TTL inputs).

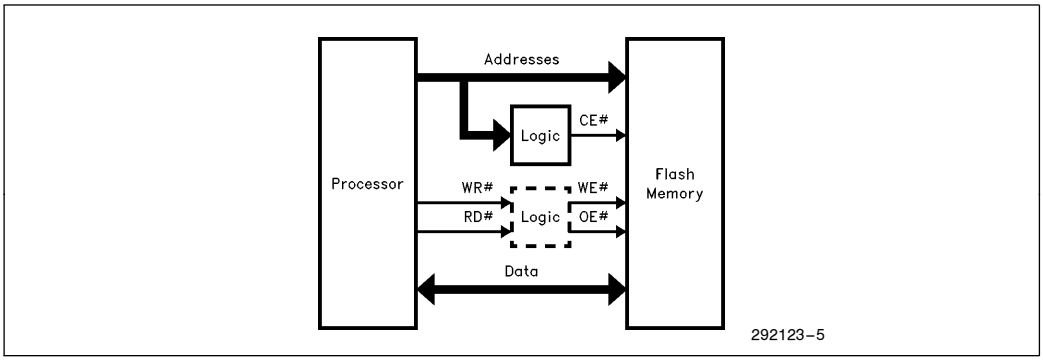

Figure 7. Basic Processor/Flash Memory Interface

- If both the V_{CC} and V_{PP} power supplies are switched "on" at the same time, one or the other is likely to ramp to a "valid" level first, depending on the relative capacitive loading at the supply outputs. Similarly, one supply will often ramp below its valid voltage range before the other, on system poweroff. This situation is acceptable, as long as the WE# and/or CE# signals to the flash memory are controlled.

Figure 7 shows a very basic example processor/memory interface. When the system power is switched on, the processor (or logic) WE# output and logic CE# output are both at GND. Depending on the processor and logic, these outputs may not reliably stabilize until V_{CC} ramps to 4.5V. In most cases, CPU and logic outputs will smoothly follow the supply voltage up to operating levels. Any oscillations on these outputs, however, can be decoded as a valid write by the flash memory, which begins to "wake up" below 4.5V V_{CC}. Similarly, address and data processor outputs are typically undefined below operating voltage ranges. Given a x8 interface between processor and flash memory (therefore, with 256 possible combinations of data inputs), there is a finite chance that a valid command byte will be randomly generated and written to the flash memory.

If the V_{PP} power supply output is less capacitively loaded than V_{CC}, V_{PP} can ramp above V_{PPL} before V_{CC} reaches 4.5V. This can cause unintended flash memory program and erase if the correct command data values are "spuriously" written to the device.

Again referencing Figure 7, the behaviour of processor/logic CE#, WE# and address/data outputs are typically undefined once V_{CC} drops below 4.5V. If the power supply V_{PP} output is more capacitively loaded than V_{CC}, V_{PP} can remain above V_{PPL} as V_{CC} decays toward 0V. This has the potential to initiate program/erase operations in response to unintended flash memory writes.

6.1 Designing for Flash Memory System Power Sequencing Protection

Intel has taken several steps with respect to its flash memory designs to significantly minimize the possibility of an unwanted write during system powerup or powerdown. By synergizing system designs to these flash memory features, you can easily eliminate the potential for unwanted flash memory mode switching and/or data alteration.

Flash memories from Intel are guaranteed *not* to program or erase with V_{PP} below 6.5V. First generation bulk-erase devices additionally block *all* write attempts with V_{PP} below 6.5V. The implication here is clear; if possible, don't switch on V_{PP} until after the system V_{CC} is stable (on powerup), and switch off V_{PP} before the system is powered down. The V_{PP} supply itself can be switched on/off, or an inline FET switch can be installed between the power supply output and flash memory input and controlled via an I/O line from the processor or discrete logic. Figure 6 gives an example of circuitry for the former case.

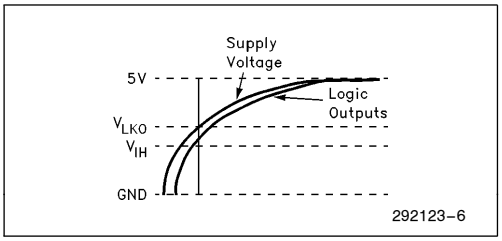

Figure 8. Supply Voltage/Device Output Relationship During Powerup

Intel flash memory also provides V_{CC}-driven "lockout protection" from unwanted writes. With V_{CC} below V_{LKO}, *all* write attempts to the flash memory are ignored. V_{LKO} varies between 2.5V and 2.0V depending on the specific flash memory, and its value is targeted to take advantage of the fact that in most cases device outputs closely follow V_{CC} inputs (both up and down). Referencing Figure 8, when V_{CC} exceeds V_{LKO}, device outputs will in most cases also be at approximately V_{LKO}, and consequently at a TTL "1" level (or disabled). The flash memory "protects itself" up to V_{LKO}, and the system designer must above that point ensure that flash memory control inputs are stable. Similarly, the flash memory is again protected once V_{CC} drops below V_{LKO} on system powerdown.

The RP# input (formerly known as PWD#), available on Intel Boot Block and FlashFile memories, acts as a "master on/off switch" for the device. With RP# at V_{IL}, the flash memory is put in a very low power mode called Deep Powerdown, and is essentially turned "off". In this state, all write attempts to the flash memory are disregarded. RP# can be driven by the POWERGOOD output of the system power supply (if this output exists) or from an external analog "power supply monitoring" device like the Maxim MAX705 or Motorola MC34064, providing absolute flash memory protection. Figure 9 gives an example system design using the Maxim component. Voltage monitoring circuits like those mentioned above have adjustable trip points and tight tolerances, and can be set to the lower value of the system logic normal operating voltage.

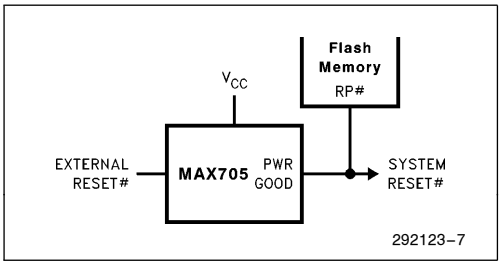

Figure 9. Reset Control during System Powerup and Powerdown

7.0 SUMMARY

Unintended writes to flash memory can, at a minimum, cause it to output data that the system does not expect, forcing system reset or power sequencing to restore normal operation. Depending on the specific data written to the device, and the V_{PP} voltage at the time of the write, actual "permanent" alteration of flash memory contents can result from unintended program or erase. However, Intel flash memory, in combination with proper system interfacing techniques, easily eliminates the potential for either of these scenarios.

Closely analyze the powerup/down and reset behaviour of the system CPU and any interface logic that interacts with the flash memory. In the vast majority of cases, no problems will be found. If potential for unwanted writes does exist, however, nonvolatile/rewriteable memory protection can easily be included if incorporated early in the design, by following the hints described in this application note.